# TP 2 Python – Fonctions et instructions conditionnelles

## 1. Indentation

Une *indentation* en Python est un décalage vers la droite de 4 espaces, obtenu en appuyant sur la touche `Tab` du clavier.

```
# Pas d'indentation ici ;
    # Une indentation au début de cette ligne ;
        # Et deux indentations ici.
```

Les indentations servent à distinguer des **blocs d'instructions**, qui sont des suites d'instruction toutes indentées, et précédées d'un en-tête. L'en-tête se termine toujours par deux points.

```
en-tête:
    Instruction1
    Instruction2
    ...
```

⚠ Ces blocs sont **omniprésents** en Python : ne jamais oublier l'indentation et les deux points !
On verra qu'on peut aussi imbriquer des blocs d'instructions les uns dans les autres.

## 2. Fonctions

**Définition**

En informatique, une fonction est un ensemble d'instructions auquel on donne un *nom*, et qu'on exécute à la demande à l'aide de ce nom. On dit alors qu'on **appelle** la fonction.
Une fonction peut :

- dépendre d'un ou plusieurs paramètres.
- renvoyer une valeur.

### 2.A) Définir une fonction en Python

```
def nom_fonction(paramètre1, paramètre2, ...):
    '''Documentation de la fonction'''
    instruction_1
    instruction_2
    ...
    return valeur_retour
```

⚠ Attention à la syntaxe ! Les deux points et l'indentation sont indispensables.
⚠ L'instruction `return` termine la fonction, même si le bloc d'instructions n'est pas fini !

**Documentation** La documentation d'une fonction est un texte qui explique clairement l'usage de la fonction. Une bonne documentation précise les paramètres de la fonction (dans l'ordre) et la valeur de retour.
On peut lire la documentation de n'importe-quelle fonction en tapant `help(nom_fonction)` dans la console.

**Exemple**

```
def présentation(nom, âge):
    '''Affichage d'une phrase de présentation.
    nom est une chaîne de caractères. âge un entier naturel.'''
    print("Bonjour ! Je m'appelle", nom, "et j'ai", âge, "ans.")
```

```python
def test():
    '''Affichage de test standard'''
    print("Hello World!")
```

```python
def exponentiation(a, n):
    y = a**n
    return y
```

Écrire une documentation pour la fonction `exponentiation`.

**Exercice 1.**
1. Écrire une fonction Python `discriminant`, qui prend en paramètre trois nombres $a, b, c \in \mathbb{R}$ et qui renvoie la valeur du discriminant du trinôme $ax^2 + bx + c$.
2. Écrire une fonction Python `hypoténuse`, qui prend en paramètre les longueurs des deux côtés adjacents à l'angle droit dans un triangle rectangle, et qui renvoie la longueur de l'hypoténuse.
   On utilisera la fonction `sqrt` du module `math`, sans oublier de l'importer.

### 2.B) Appeler une fonction en Python

Une fois la fonction `nom_fonction` définie, on peut l'utiliser en l'*appelant* avec des valeurs bien précises. On utilise simplement l'instruction `nom_fonction(valeur1, valeur2, ...)`.

**Exemple**

```
>>> exponentiation(3, 4)
81
```

```
>>> exponentiation(2.5, 2)
6.25
```

**Exercice 2.**
1. À l'aide de votre fonction `discriminant`, déterminer le nombre de solutions réelles de l'équation $3x^2 - 6,08x + \pi$.
2. Testez votre fonction `hypoténuse` en déterminant l'hypoténuse d'un triangle rectangle dont les deux petits côtés ont pour longueur 5 et 12.

## 3. Instructions conditionnelles

### 3.A) Booléens et tests

**Définition**

Le type **booléen** (`bool`) est constitué des deux valeurs logiques : `True` (Vrai) et `False` (Faux).

Un **test logique** est une procédure qui vérifie la vérité d'un énoncé. Le résultat est une valeur booléenne (True si l'énoncé et vrai et False si il est faux).
On peut appliquer les **opérations logiques habituelles** en Python. Ainsi, si $a$ et $b$ sont deux booléens :

- `not a` renvoie le booléen associé à non $a$
- `a or b` renvoie le booléen associé à $a$ ou $b$
- `a and b` renvoie le booléen associé à $a$ et $b$

On peut également **comparer des nombres**, avec les symboles suivants :

- Test d'égalité : symbole `==`
- Test de différence : symbole `!=`
- Test d'ordre strict : symboles `<` et `>`
- Test d'ordre large : symboles `<=` et `>=`

⚠ Le symbole « = » a une autre signification, il sert à affecter des variables. N'oubliez pas le « double égal » (« == ») quand vous testez une égalité, c'est une erreur très classique !

**Exemple**

```
>>> 1 == 5 - 2
False
```

```
>>> 3 != -3
True
```

```
>>> 2 >= 3
False
```

```
>>> (1 == 3 - 2) and (1 < 0)
False
```

**Exercice 3.** Déterminer la valeur des booléens suivants :

a) `2*2 == 2**2` b) `(12 < 3*4) or (9 >= 3*3)` c) `(6%2 == 0) and (6%3 != 0)`

⚠ Ne **jamais** effectuer de test d'égalité sur des flottants ! Les erreurs d'arrondis ne garantissent pas l'égalité. Si nécessaire, il vaut mieux tester si leur différence est inférieure à la précision recherchée. Par exemple :

```
>>> 0.1+0.1+0.1 == 0.3
False
```

```
>>> 0.1 + 0.1 + 0.1 - 0.3
5.551115123125783e-17
```

```
>>> (0.1+0.1+0.1)-0.3 < 10**(-6)
True
```

## 3.B) Instructions conditionnelles en Python

Pour écrire des instructions conditionnelles, qui dépendent d'une certaine condition logique, Python dispose de trois mots-clés à connaître : `if`, `else` et `elif`.
La structure de base d'une instruction conditionnelle est la suivante :

```
if condition :
    bloc_instructions
else :
    bloc_alternatif
```

Python commence par déterminer la valeur logique de `condition`. Si elle est vraie, il exécute les instructions de `bloc_instructions`. Sinon, il exécute les instructions de `bloc_alternatif`.

*Remarque.* ⚠ Syntaxe : ne pas oublier les deux points et l'indentation.
Il n'est pas obligatoire de définir des instructions alternatives. Dans ce cas, on se contentera d'écrire :

```
if condition :
    bloc_instructions
```

Si la condition n'est pas vérifiée, il ne se passe rien.

**Avec plusieurs alternatives** Python dispose du mot clé `elif` pour exécuter un bloc d'instruction lorsqu'une première condition n'est pas vérifiée mais qu'une seconde l'est. La structure est alors la suivante :

```
if condition_1 :
    bloc_instructions_1
elif condition_2 :
    bloc_instructions_2
else :
    bloc_instructions_3
```

*Remarque.*

- `elif` est une abréviation de "else if"
- Le "else" n'est toujours pas obligatoire (dans ce cas, si aucune condition n'est vérifiée, il ne se passe rien).
- On peut traiter encore plus de cas en ajoutant plus de « elif ».

**Exemple**

Le prix d'un trajet est de 10€ pour un seul passager, 8€ par personne pour 2 à 3 passagers, 7€ par personnes pour 4 à 6 passagers et 6€ par personne à partir du septième passager.
En Python, le calcul du prix total pour peut s'écrire :

```
def prix_total(nb_pass):
    '''nb_pass est le nombre de passagers. La fonction renvoie le prix total du
    trajet pour ce nombre de passagers.'''
    if nb_pass == 1 :
        return 10
    elif nb_pass == 2 or nb_pass == 3 :
        return 8 * nb_pass
    elif nb_pass <= 6 :
        return 7 * nb_pass
    else :
```

```
        return 6 * nb_pass
```

**Exercice 4.** Que fait la fonction suivante ?

```
def mystère(x) :
    if x >= 0 :
        return x
    else :
        return -x
```

**Exercice 5.** Sur l'autoroute, on ne peut pas rouler à moins de 80 km/h sur la voie de gauche. Et bien sûr, on ne peut jamais rouleur à plus de 130 km/h !
Écrire une fonction `radar(vitesse)`, qui prend en argument une vitesse en km/h et qui affiche, en fonction des cas :

- "Vous êtes en infraction !"
- "Vous ne pouvez pas rouler sur la voie de gauche de l'autoroute."
- "C'est bien, continuez."

### 3.C) Diagrammes de flux

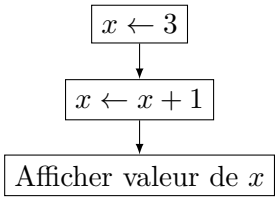
On représente parfois par un diagramme l'enchaînement (le flux) des instructions exécutées par un algorithme. On appelle ce genre de schéma un **diagramme de flux**, et on respecte les conventions suivantes :

- Des flèches représentent l'ordre de déroulement des instructions.
- Une instruction dans un rectangle n'a qu'une seule suite possible.
- Une instruction dans un losange a plusieurs suites possibles.

*Exemple :* suite de 3 instructions en python et en pseudo-code, ainsi que le diagramme de flux associé :

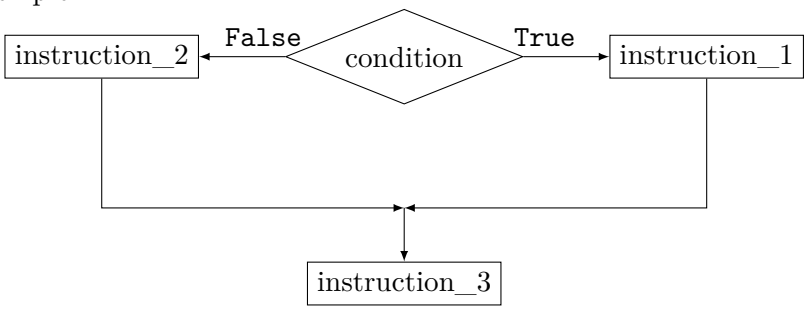
```
x = 3
x = x + 1
print(x)
```

$x \leftarrow 3$
$x \leftarrow x + 1$
Afficher valeur de $x$




Les instructions conditionnelles permettent de réaliser des branchements dans le diagramme de flux, en fonction d'une condition logique. Par exemple :

```
if condition :
    instructions_1
else :
    instruction_2
instruction_3
```




**Exercice 6.** Dessiner le diagramme de flux de la fonction Python suivante, puis déterminer sa valeur pour $x = -1$, $x = 0$, $x = 2$ et $x = 4$.

```
def fonction(x):
    a = 5
    if x < 0 :
        a = a - x
    elif x <= 3 :
        a = a * x
    else :
        a = a + x
    b = a ** 2
    return b
```

## 4. Exercices bilan

**Exercice 7.** On suppose qu'une variable `a` a été préalablement initialisée.
Expliquer la différence entre les deux programmes suivants et donner une valeur de `a` pour laquelle les deux programmes attribueront en fin d'exécution une valeur différente à `b` :

```
b = 1
if a > 3:
    b = 2*a
b = b+1
```

```
b = 1
if a > 3:
    b = 2*a
    b = b+1
```

**Exercice 8.** Définir les fonctions Python suivantes :

1. `différence(x,y)`, qui prend en paramètres deux nombres réels et qui renvoie leur différence.
2. `implique(A,B)`, qui prend en argument deux booléens `A` et `B` et renvoie la valeur logique de l'implication $A \Longrightarrow B$.
3. `fonction(x)`, qui à tout nombre réel $x$ entré en paramètre, renvoie : $\begin{cases} 2x+1 & \text{si } x \geq 4 \\ 2x & \text{si } x \in [2;4[ \\ 3-2x & \text{sinon} \end{cases}$
4. `est_pair(n)`, qui prend en entrée un entier $n$, et qui renvoie le booléen `True` si $n$ est pair, et le booléen `False` sinon.

**Exercice 9.** Soient trois variables `a`, `b` et `c` dont les valeurs sont les coefficients réels d'un trinôme $ax^2+bx+c$. Écrire un stript Python qui détermine le nombre de racines (réelles) du trinôme, ainsi que leur valeur.

**Exercice 10.** Selon Wikipédia : « Les années sont bissextiles si elles sont multiples de quatre, mais pas si elles sont multiples de cent, à l'exception des années multiples de quatre cents qui, elles, sont également bissextiles. ».

1. Écrire une fonction `bissextile(année)` qui prend en paramètre un entier (une année), et **affiche** une phrase qui dit si l'année est bissextile ou non.
2. Testez votre fonction avec les années suivantes : 2000, 2007, 2024, 2026, 2100.

**Exercice 11.** Soient $A$ et $B$ deux variables de type booléen. À l'aide d'instructions conditionnelles, écrire des programmes qui attribuent à une variable $C$ la valeur booléenne :

1. de la *négation* de $B$, *sans utiliser* l'instruction `not`.
2. du *et logique* de $A$ et $B$, *sans utiliser* l'instruction `and`.
3. du *ou logique* de $A$ et $B$, *sans utiliser* l'instruction `or`.